Erin Jones
CE-263H Fall 2023
Assignment 3 Part 2

Processing Individual Data

1) How many points does your selected trajectory have?

```
{'filter': {'function': 'filter', 'max_speed_kmh': 200.0, 'include_loops': True, 'speed_kmh': 5.0, 'max_loop': 6, 'ratio_max': 1}}
Points of the raw trajectory:          2160
Points of the filtered trajectory:     2012
Filtered points:                       148
```

2) Is the recording of the google locations continuous (i.e. reported in a continuous set of time steps)?

No! See the notebook, but I wrote some code to check for consecutive timestamps and the timestamps are not consecutive.

3) Cluster stops

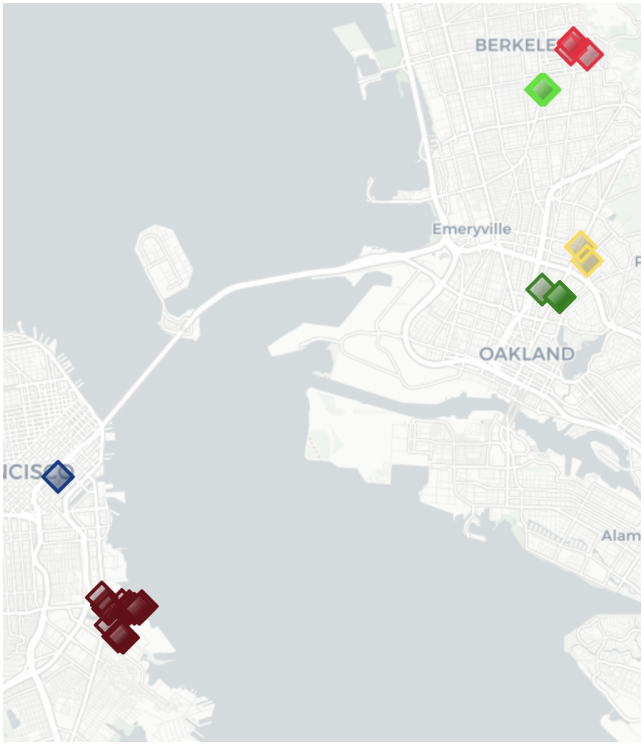
a) Parameters Defined

```
# default stop radius factor
stdf_1 = detection.stay_locations(ctdf, stop_radius_factor=0.5, minutes_for_a_stop=15.0, spatial_radius_km=0.1, leaving_time=True)
print(stdf_1.parameters)
print('Points of the original trajectory:\t%s'%len(tdf))
print('Points of stops:\t\t\t%s\n'%len(stdf_1))

# 'zooming out' per se
stdf_2 = detection.stay_locations(ctdf, stop_radius_factor=1.0, minutes_for_a_stop=60.0, spatial_radius_km=1.5, leaving_time=True)
print(stdf_2.parameters)
print('Points of the original trajectory:\t%s'%len(tdf))
print('Points of stops:\t\t\t%s'%len(stdf_2))
```

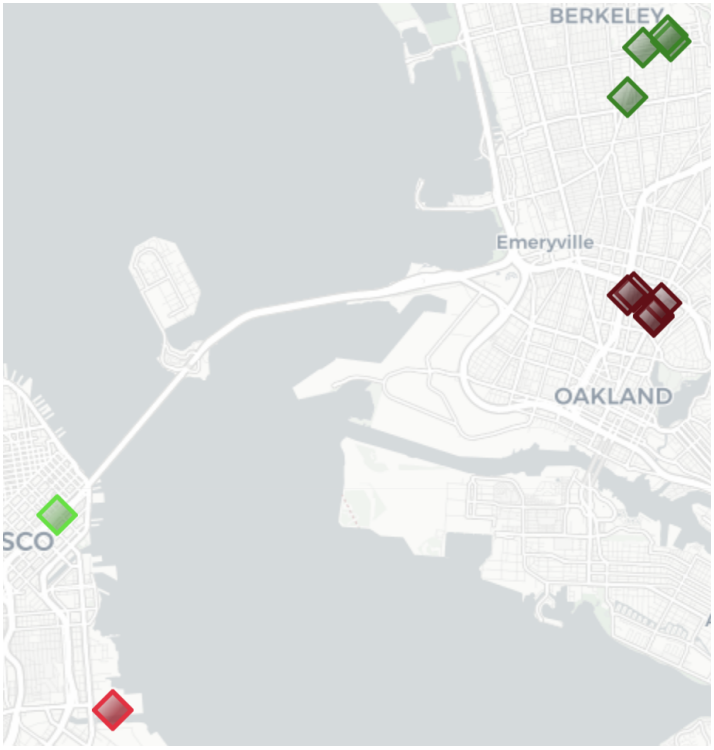
```
{'filter': {'function': 'filter', 'max_speed_kmh': 200.0, 'include_loops': True, 'speed_kmh': 5.0, 'max_loop': 6, 'ratio_max': 1},
Points of the original trajectory:      2160
Points of stops:                        35

{'filter': {'function': 'filter', 'max_speed_kmh': 200.0, 'include_loops': True, 'speed_kmh': 5.0, 'max_loop': 6, 'ratio_max': 1},
Points of the original trajectory:      2160
Points of stops:                        12
```

b) Cluster Map

i) Parameters 1

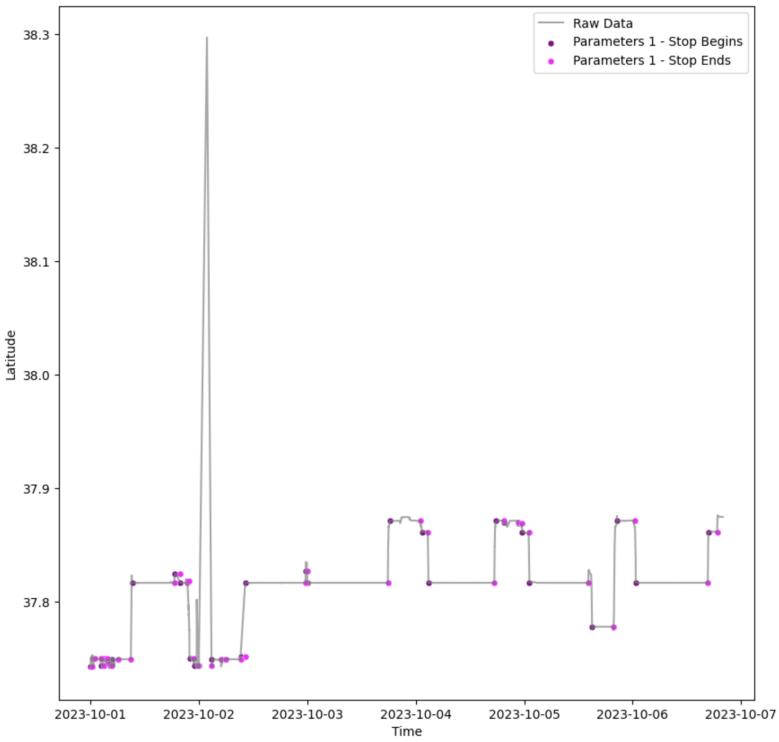


ii) Parameters 2

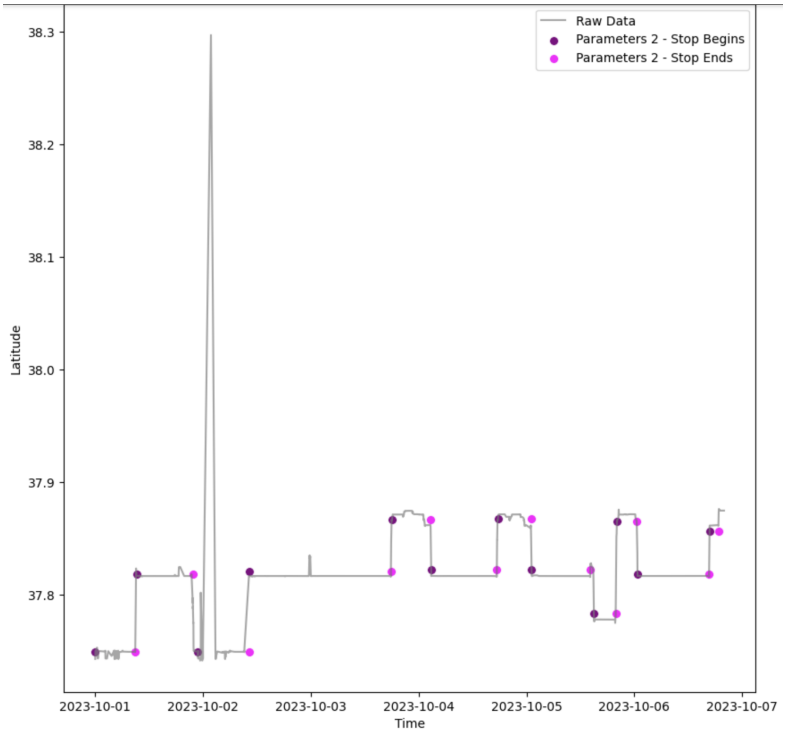


c) Lat v Time

i) Parameters 1



ii) Parameters 2



4) Which set of parameters is better?

I think that each set of parameters serves a different purpose – in essence it depends on how granular we want the log of our movements to be. In the first set of parameters, I capture lots of details, like a stop in a specific class or, during this weekend I was at a music festival in the city and can even see stops at the different stages on the pier. This would be good for building a very specific diary of movement with the caveat that sometimes this type of location data is not great at pinpointing location to a tee, so there may be hiccups.

The second technique, I am making bigger holes in the net and only capturing larger fish, to use an allegory. I will only catch those stops where I am in a location for an extended period of time. This works well for brushing over some of the noise of say moving around within a festival or on campus. It seems to work when looking just at the stops detected.

Now - the parameters start to make a big difference when we begin to look at clustering. The first technique accurately separates activities like going to the gym (on south shattuck) from going to campus. The 'goodness' of this decision depends on what behavior we are trying to describe though. I often will go to the gym and to campus in one go, because it is most time efficient, but they really aren't the same thing and one can't consider 100% of my time as being spent doing school stuff. The second technique cleans out the noise and we end up with several sparsely populated clusters in the city. It does seem to erase a lot of my behavior, but it still captures a good high level overview of what was occuring that weekend.

5) Unique Destinations

Erin Jones
CE-263H Fall 2023
Assignment 3 Part 2

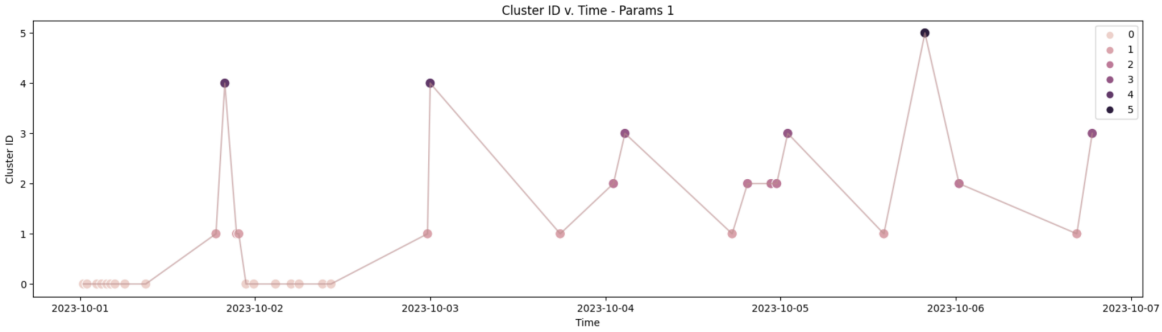
Question 5 - How many unique destinations were visited in the selected period?

```
stdf_2['lat_lng'] = stdf_2['lat'].astype(str) + ", " + stdf_2['lng'].astype(str)
stdf_2['lat_lng'].value_counts()
```

```
37.74927535, -122.3824775          1
37.8181546, -122.26231089999999    1
37.7493235, -122.3824775           1
37.820738, -122.2607718            1
37.86654475, -122.25852545000001   1
37.8225345, -122.2670148           1
37.867471625, -122.25927025        1
37.821938450000005, -122.2683802   1
37.7833434, -122.3949012           1
37.8654609, -122.26462190000001    1
37.8184694, -122.2624707           1
37.8566199, -122.268229075         1
Name: lat_lng, dtype: int64
```

This method *technically* identifies 12 stops at 12 unique geographic coordinates. At the precision accuracy of 3 decimal places in a lat/long coordinate, the error is ~100m. At 4 decimal places, the error is ~10m (https://support.garmin.com/en-US/?faq=hRMBoCTy5a7HqVkxukhHd8). That said, I would argue that the first 2 should probably be considered a single location. Actually...there are probably quite a few on this list that should be considered single locations. The parameters themselves extract 12.

6) Cluster ID v Time



7) Home v Work v Other - see notebook for graphs

In the 1st set of parameters

- **0** - Zero is Portola Festival on Pier 80
- **1** - One is home in this set
- **2** - Parking garage and campus
- **3** - Is the gym on south shattuck.
- **4** - Is a meal I had in Piedmont and then a weird anomalous record maybe where I grabbed food? This makes me think that maybe the timestamps are off by a timezone and I need to convert them to pacific, because I wouldn't have gotten anything up Broadway after 11pm on a weeknight.
- **5** - Is my internship in SOMA

Again...the 'best' set of parameters really depends on your goal. I think the second set does a better job summarizing into high level 'home' 'work' and 'other' broad band clusters. The first one, while it does highlight the activity at pier 80 a little too much, which is actually a one off event and not habitual, does separate out grabbing food or going to the gym, which better summarizes my behavior.

In the 2nd set of parameters

- **0** - Zero represents home. 37.816764, -122.263268 are the actual coordinates of my home and it lines up with those clustered in that region
- **1** - Cluster 1 is school, but my stop at the gym on south shattuck is also lumped in there
- **2** - Two is Portola festival at pier 80
- **3** - Is my internship in SOMA

8) Select three individual measures - implement and explain

a) Home Location

Home Location

```
[ ] from skmob.measures.individual import home_location

    hl_df = home_location(ftdf)
    hl_df.head()
```

	lat	lng
0	37.816606	-122.262396

Hooray!!! It nailed it! This is indeed my home location, and I even used the default values of 22hr for 'coming home for the evening' and 7hr for the hour of 'leaving in the morning'. In essence this formula finds the location where the probability metric calculated is greatest (argmax). The probability being calculated is the probability that individual u (me in this case) is at location r (specified in lat/long coordinates) given a specific time of day, where the algorithm only assesses times that fall between 22hr (10pm) and 7hr (7am) to make an educated guess that the location generating the highest conditional probability under these circumstances is likely the person's home location. The parameters for start_night and end_night which bind the assumption that the individual is home between those hours, are modifiable. This would be necessary in the event that I worked a night shift or something like that.

b) Frequency Ranking

Frequency Rank

Frequency rank a simple, sorted frequency distribution based on how frequently an individual visits a location. The plumbing behind this formula is very simple - it groups the dataframe by lat/long and then pulls value counts for the number of occurences associated with a given location. This frequency distribution is then sorted in descending order and given the frequency rank index once sorted.

```
from skmob.measures.individual import frequency_rank

fr_df = frequency_rank(ftdf)
fr_df.head()
```

	lat	lng	frequency_rank
0	46.892265	-113.955658	1
1	46.892266	-113.955662	2
2	46.892289	-113.955654	3
3	46.892279	-113.955660	4
4	46.892265	-113.955664	5

I really want to get a better feel without the noise of the high precision lat/long coordinates, which more than likely are incorrect. I am going to round at 5 decimal places and see where that puts the results for frequency rank.

```
df = ftdf.copy()
df['lat'] = df['lat'].round(5)
df['lng'] = df['lng'].round(5)

fr_df_1 = frequency_rank(df)
fr_df_1.head()
```

	lat	lng	frequency_rank
0	46.89227	-113.95566	1
1	37.81661	-122.26240	2
2	37.81663	-122.26235	3
3	37.81662	-122.26240	4
4	37.81661	-122.26241	5

FASCINATING!!! This *completely* changes the results. I did a bit of googling to see where the heck each of these locations fell. The top dataframe is entirely dominated by the week or so that I spent back in Montana as the top 5 locations are basically my childhood home. When I rounded the lat and long columns to 5 decimal places, ranks 2-5 are replaced by my apartment in Oakland. Which is really bizarre because I was only in Montana for a week.

This essentially should create a list of my most visited locations. The garage behind my apartment makes sense. My house in Montana does not. It makes me wonder about my location permissions, when I had my device on, etc.

c) Radius of gyration

```
[ ] from skmob.measures.individual import radius_of_gyration

    rg_df = radius_of_gyration(ftdf)
    rg_df.head()
```

	radius_of_gyration
0	314.817259

Radius of Gyration

The above calculated metric represents the characteristic distance traveled by me up to time t=08/10/2023 in kilometers [1]. This drops significantly though when I remove the period that includes my trip to Montana.

Just summarizing the formula for radius of gyration takes the squareroot of the sum (across all positions in the trajectory dataframe for the individual) of an individuals position, minus the center of mass of the trajectory squared.

In looking further into the utils files behind skmob, the function itself (which works on an array of trajectories for many individuals or on the trajectory of a single individual) first creates an array storying the lat_long pairing from all the points making up an individuals trajectory. The mean of this array (a scalar value) is then calculated and stored as an individuals center of mass. Then, skmob calls a utils function that will calculate the distance of each location denoted by lat long coordinates in the trajectory frame, from the center of mas, via the Haversine distance formula. These distances are squared and then the average is taken (they are all summed, then divided by the number of locations) and finally the squareroot is taken on the result of the np.average.

In this sense, it makes sense that my radius of gyration would be large if I took a big trip during the time period under investigation. The locations recorded in that 10 day period were very very far from the general points that make up the trajectory and my center of mass. We can see this is the case below when the period including my trip is trimmed out and the radius drops to 8km.

```
[ ] subframe = (ftdf[(ftdf['datetime'] > '2023-07-04') & (ftdf['datetime'] < '2023-08-04')]).copy()
    rg_df_1 = radius_of_gyration(subframe)
    rg_df_1.head()
```

	radius_of_gyration
0	8.304369

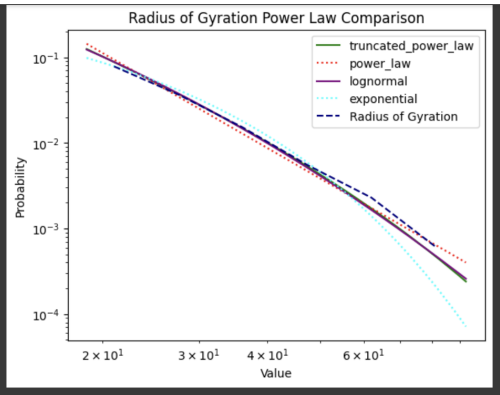
9) Fitting Power Laws

The major difference between the MassDOT survey and the cell phone survey data has to do with the method of data collection. MassDOT in 2010 included a subsample of 500 households that used GPS to track their mobility and as far as I can tell contains no cell phone data whatsoever. The method of location detection between GPS and cell phone is slightly different and the differences between the two are compared thoroughly by Pappalardo and Simini in their paper “Data-driven generation of spatio-temporal routines in human mobility” [2]. The GPS data is transmitted more frequently and regularly than the CDR type data and we expect this MassDOT data to align more closely with the patterns seen in the Pappalardo and Simini paper for the same analysis.

In reality, the radius of gyration seems to mimic the shape of the CDR data more closely as seen in Figure 2 of [2]. The duration of stay appears to be more similar to the GPS data seen in Figure 7 of [2].

a) Radius of Gyration

[69]



Based on the above, it is hard to say which estimate is the best suited. The data maps closely with the lognormal and truncated power law functions for smaller values, but then the trajectory as x gets big align more nicely with the standard power law. The parameters derived by the power law package are printed below.

Given the r value generated by distribution comparison below, the truncated power law wins out.

```
[70] radius_dict = comp_dist(fit_rg)
     radius_dict
```

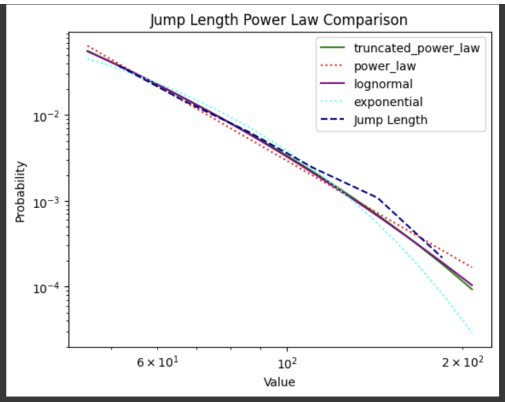
```
Assuming nested distributions
{('truncated_power_law', 'power_law'): (16.929912630769216,
  5.92292148600785e-09),
 ('truncated_power_law', 'lognormal'): (3.06100415992773,
  2.0469830862149836-07),
 ('truncated_power_law', 'exponential'): (28.263203212414524,
  3.22321204775665956-05),
 ('power_law', 'lognormal'): (-13.868908470842001, 0.00046540305486801256),
 ('power_law', 'exponential'): (11.333290581645162, 0.2890863475483235),
 ('lognormal', 'exponential'): (25.202199052487245, 0.0002204125464932953)}
```

```
[71] print(f"For the truncated power law equation, alpha is calculated as: {tpl_alpha}\n")
     print(f"For the truncated power law equation, alpha is calculated as: {tpl_lambda}\n")
```

```
For the truncated power law equation, alpha is calculated as: 2.3531285169767733

For the truncated power law equation, alpha is calculated as: 0.0339379916929703
```

Erin Jones
CE-263H Fall 2023
Assignment 3 Part 2

b) Jump Length



In the above, none of the fits seem to do a great job due to the weird elbow that occurs in the right-hand side of the graph. Again, the truncated power law wins out this time

The parameters are printed below.

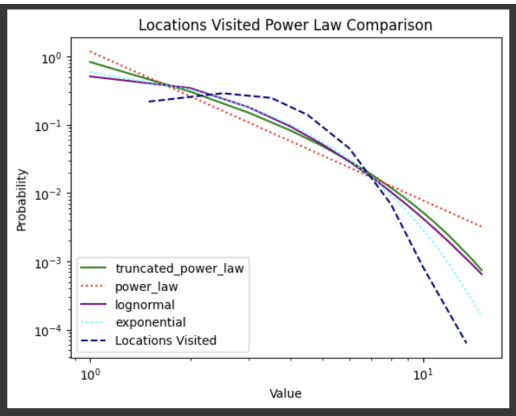
```
jump_dict = comp_dist(fit_jl)
jump_dict
```

```
Assuming nested distributions
{('truncated_power_law', 'power_law'): (15.201410146087525,
  3.5101744733090356e-08),
 ('truncated_power_law', 'lognormal'): (3.1586351195892295,
  3.0085584631302276e-09),
 ('truncated_power_law', 'exponential'): (20.10033218830441 7,
  0.0001748862908632934),
 ('power_law', 'lognormal'): (-12.042775026498118, 0.0007366318562617019),
 ('power_law', 'exponential'): (4.898922042217034, 0.5848942545163756),
 ('lognormal', 'exponential'): (16.941697068715346, 0.0019446068047508055)}
```

```
[77] print(f"For the truncated power law equation, alpha is calculated as: {tpl_alpha}\n")
     print(f"For the truncated power law equation, alpha is calculated as: {tpl_lambda}\n")
```

```
For the truncated power law equation, alpha is calculated as: 2.380742301482421

For the truncated power law equation, alpha is calculated as: 0.017220143080866203
```

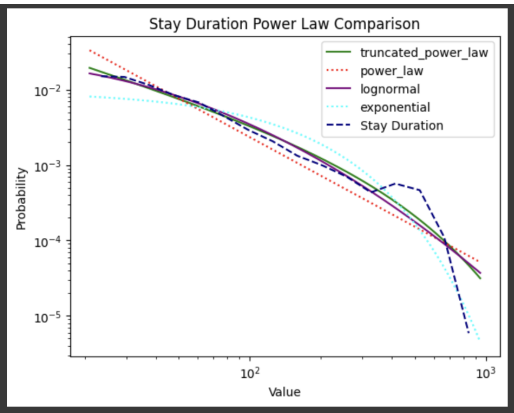
c) Locations Visited



They all do a *terrible* job in this scenario. The exponential has the largest curve to it and thus seems to mimic the shape in a more apt fashion. This is proven by the distribution comparison results below

```
[79] vl_dict = comp_dist(fit_vl)
     vl_dict

     Assuming nested distributions
     {('truncated_power_law', 'power_law'): (3961.259677238608, 0.0),
      ('truncated_power_law', 'lognormal'): (184.32582006523035,
       6.04844774337802e-05),
      ('truncated_power_law', 'exponential'): (-928.7643375086695,
       4.279738330065567e-141),
      ('power_law', 'lognormal'): (-3776.9338571752155, 0.0),
      ('power_law', 'exponential'): (-4890.024014747797, 0.0),
      ('lognormal', 'exponential'): (-1113.0901575751245, 0.0)}

[80] print(f"For the exponential equation, lambda is calculated as: {e_lambda}\n")

     For the exponential equation, lambda is calculated as: 0.5900583814188354
```

d) Duration of Stay



There is a rather large hiccup on the righthand side of the graph, but up until that point it seems to follow the lognormal and truncated powerlaw functions. After a point it seems to dive to meet the exponential function. It seems you may be able to define the relationship piecewise such that in between a particular range of values, one function best describes the relationship and after that point we switch to a different function.

Based on the distribution comparison, the truncated power law wins out. The coefficients are included below.

```
[82] wt_dict = comp_dist(fit_wt)
     wt_dict

     Assuming nested distributions
     {('truncated_power_law', 'power_law'): (7288.437839003373, 0.0),
      ('truncated_power_law', 'lognormal'): (1219.6367975684777, 0.0),
      ('truncated_power_law', 'exponential'): (6363.176055034709, 0.0),
      ('power_law', 'lognormal'): (-6068.801041434, 0.0),
      ('power_law', 'exponential'): (-925.2617839670372, 1.3971789685602981e-07),
      ('lognormal', 'exponential'): (5143.539257467438, 0.0)}

[83] print(f"For the truncated power law equation, alpha is calculated as: {tpl_alpha}\n")
     print(f"For the truncated power law equation, alpha is calculated as: {tpl_lambda}\n")

     For the truncated power law equation, alpha is calculated as: 1.000000058771401

     For the truncated power law equation, alpha is calculated as: 0.002840189550307123
```

References

[1] González, M., Hidalgo, C. & Barabási, AL. Understanding individual human mobility patterns. Nature 453, 779–782 (2008). https://doi.org/10.1038/nature06958

[2] Pappalardo, L., Simini, F. Data-driven generation of spatio-temporal routines in human mobility. Data Min Knowl Disc 32, 787–829 (2018). https://doi.org/10.1007/s10618-017-0548-4